DLOGTIME Proof Systems

Andreas Krebs[1] and Nutan Limaye[2]

1 University of Tübingen, Germany. mail@krebs-net.de
2 Indian Institute of Technology, Bombay, India. nutan@cse.iitb.ac.in

Abstract

We define DLOGTIME proof systems, DLTPS, which generalize NC^0 proof systems. It is known that functions such as Exact_k and Majority do not have NC^0 proof systems. Here, we give a DLTPS for Exact_k (and therefore for Majority) and also for other natural functions such as Reach and Clique_k. Though many interesting functions have DLTPS, we show that there are languages in NP which do not have DLTPS. We consider the closure properties of DLTPS and prove that they are closed under union and concatenation but are not closed under intersection and complement. Finally, we consider a hierarchy of polylogarithmic time proof systems and show that the hierarchy is strict.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases Proof systems, DLOGTIME, NC^0

1 Introduction

In a seminal paper by Cook and Reckhow [6] a *proof system* for a language was defined as a polynomial time computable function, called a *verifier*, whose range is all the words in the language. For any language in NP, there exists such a proof system. This therefore gives a way of viewing the complexity classes in the framework of proof systems. Polynomial time verifiers naturally characterize languages in NP. To understand complexity classes that contain NP, some works have considered proof systems with more powerful verifiers. (See for example [9, 5, 4, 3].)

Taking a dual approach, proof systems with very weak verifiers have been recently studied in [2, 8]. It is known that for any language in NP there exists a uniform AC^0 proof system, i.e. a proof system in which the verifier is a function computable by a uniform AC^0 circuit. In [2], a restriction of AC^0 proof systems, namely $\mathrm{NC}^0\mathrm{PS}$ was considered, where the verifier is a (possibly non-uniform) NC^0 circuit. They observed that there are NP complete languages which have $\mathrm{NC}^0\mathrm{PS}$. However, there are even regular languages for which there are no $\mathrm{NC}^0\mathrm{PS}$. It is natural to define a proof system that generalizes $\mathrm{NC}^0\mathrm{PS}$, but is not as general as proof systems for NP.

In this work we investigate a proof system which generalizes uniform $\mathrm{NC}^0\mathrm{PS}$ (i.e. the verifier is a uniform NC^0 circuit) but is more restrictive than $\mathrm{AC}^0\mathrm{PS}$. We consider a proof system in which the verifier is a deterministic log-time Turing machine.

Deterministic log-time Turing machines can compute an AND of $\omega(1)$ bits and therefore are more powerful than uniform NC^0 circuits. However, they cannot compute an AND of $\Theta(n)$ bits, and therefore are less powerful than uniform AC^0 circuits. Also NC^0 verifiers can make only $O(1)$ queries to the *proof bits*[1] as opposed to DLOGTIME verifiers which can make $O(\log n)$ queries. (See Section 2 for formal definitions.) We see that this makes DLTPS much more powerful than uniform $\mathrm{NC}^0\mathrm{PS}$. Note, however, that the DLOGTIME verifier is

[1] The input to the verifier is called a proof and it is a string over some fixed alphabet

Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

by default uniform. At first, this may seem like a big restriction. However, it is interesting to note that most languages which are shown to have NC^0 proof systems in [2], in fact have uniform proof systems.

Our results:

- In the same spirit as in [2, 8], we prove that many interesting natural functions have DLTPS. In [2], it was proved that functions such as ExactOR, Majority, Exact_k do not have NC^0PS. We prove (Section 3) that for any DLOGTIME computable function $k : \mathbb{N} \to \mathbb{N}$, Exact_k has DLTPS, and hence ExactOR and Majority also have DLTPS.
- We consider some well-known graph problems. We prove that Reachability on directed graphs has DLTPS. We also prove that for any DLOGTIME computable function $k : \mathbb{N} \to \mathbb{N}$, Clique_k has a DLTPS (Section 4).
- As a part of our study of DLTPS we analyze the closure properties of DLTPS (Section 5). We prove that it is closed under union and concatenation. That is, if L_1 and L_2 are two languages which have DLTPS then $L_1 \cup L_2$ and L_1L_2 also have DLTPS. On the other hand, we prove that DLTPS are not closed under intersection and complement.
- We prove that there is a language in NP for which there is no DLTPS. We prove this by showing that any language which has DLTPS can be recognized by a non-deterministic Turing machine in time $O(n^2 \log^3 n)$ (Theorem 10). A similar lemma was proved in [2] for NC^0PS. Our proof is similar in spirit, however we have to be more careful in our argument due to the adaptive nature of the queries made by the DLOGTIME verifier.
- Finally, we consider a hierarchy of polylogarithmic time proof systems. We show that this hierarchy is strict (Section 6). Our proof of the hierarchy theorem uses lazy diagonalization as in the proof of non-deterministic time hierarchy theorem [10]. However, our proof is more delicate due to some technical reasons: there are two crucial parameters in our hierarchy theorem– the proof length and the running time of the DLOGTIME machine. A naive implementation of non-deterministic time hierarchy theorem gives a hierarchy on these two parameters simultaneously. However, we observe that DLOGTIME machine cannot access too many proof bits and that diagonalization does not need to compute the full output of the proof system, but only the length of the output. We believe that our proof may be of independent interest.

Related work: Krebs et al. in [8] defined poly log AC^0 proof systems. The verifier for this proof system is a bounded fan-in $O(\log\log n)$ depth circuit (possibly non-uniform) with at most $O(1)$ alternations. By definition, poly log AC^0 proof systems generalize NC^0PS and are a restriction of AC^0 proof systems.

Though DLTPS and poly log AC^0 proof systems are both generalizations of NC^0PS, they both are very different. A poly log AC^0 verifier of depth $c \log\log n$ can query $O(\log^c n)$ proof bits; whereas a DLOGTIME verifier can query at most $O(\log n)$ bits. In this sense, poly log AC^0 proof system is more powerful than DLTPS. However, a DLOGTIME verifier can make adaptive queries to the proof bits[2]. This means that for computing one output bit, a DLOGTIME verifier can potentially query all proof bits.

Interestingly, [8] gave poly log depth AC^0 proof systems for Exact_k, ExactOR, and Majority, but the the DLTPS and poly log AC^0 proof systems for these functions are fundamentally different. .

[2] The queries made to the proof bits are said to be adaptive if having read a few bits of the proof, the locations of the bits to be read subsequently depend on the values of those bits.

2 Proof Systems

DLOGTIME Turing machines have been studied in the past. (See for example [1, 7].) We use the definition from [1]. A DLOGTIME Turing machine is a deterministic Turing machine which has an input tape, a constant number of read-write tapes, and an index tape. We assume that none of the tapes have an end marker. The machine runs in time $O(\log n)$ time, where n is the length of the input. In one step, the machine can read a bit from the input indexed by the index tape.

▶ **Definition 1.** A function $f : \Gamma^* \to \Sigma^*$ is computable in DLT if there exists a DLOGTIME Turing machine M such that $M(\sigma, i, x)$ halts and accepts iff $w_i = \sigma$, where $w_0 \dots w_{|x|-1} = f(x)$.

▶ Remark. Not every DLT describes a function. Consider a machine M that accepts no letter at the first position but a letter at the second position.

We assume that the index i is given in binary notation. In general, whether i is encoded in binary or unary is not important to the power of the functions considered here, since we can always assume that $i \leq |x|$.

A language is said to be accepted by a Turing machine if it accepts all the words in the language and nothing else. For proof systems, this notion is reversed. A language is said to have a Turing machine as its proof system if the output of the machine is exactly all the words of the language, while the input ranges over all possible strings. Formally,

▶ **Definition 2** (Proof System)**.** Let Σ, Γ be alphabets. A proof system for $L \subseteq \Sigma^*$ is a map $f : \Gamma^* \to L$, that is onto. A proof system f is polynomial bounded if there is a polynomial p such that for every $y \in L$ there exists an x with $f(x) = y$ and $|x| < p(|y|)$.

▶ Remark. Our proof systems cannot accept the empty language, i.e. no word at all. The "smallest" language we can accept is the language which contains only the empty word, by an oracle that always rejects. The input alphabet can be assumed to be $\{0, 1\}$ without loss of generality (by binary encoding of Γ).

In the definition there are two properties required for f to be a proof system of L. First, for all inputs x the output $f(x)$ must be in L. We refer to this property as *correctness*. Second, for every $y \in L$ there is an input x such that $f(x) = y$, i.e. f is surjective.

Here we will study proof systems where the function f is computable in DLT. Given that a machine that runs in logarithmic time cannot output a long string, we say that a function f is computable in DLT if every bit (or letter) of the output is computable in DLT.

Combining the last two definitions we say the language L has a DLT proof system (DLTPS) if there is a polynomial bounded proof system f for L such that f is computable in DLT. Note that, as input and output lengths in DLTPS are polynomially related, and as the computational power of DLTPS is bounded by logtime in terms of the input length, it is also bounded by logtime in terms of the output length.

In [1], simple functions were shown to be computable in DLOGTIME. They showed that given an input x, a DLOGTIME machine can compute $|x|$ by a double binary search. Addition and subtraction of two $O(\log n)$ bit numbers can be done in DLOGTIME. Logarithm of a $O(\log n)$ bit number can be computed in DLOGTIME.

3 Word Problems

In this section we give DLTPS for various languages which are subsets of $\{0, 1\}^*$. ExactOR is a set of all string from $\{0, 1\}^*$ with exactly one bit set to 1. For $k : \mathbb{N} \to \mathbb{N}$, $\text{Exact}_k \subset \{0, 1\}^*$, is a set of all strings with exactly k bits set to 1. And Majority $\subset \{0,1\}^*$ is a set of all strings with at least as many 1s as 0s.

In [2], the above functions were considered. They proved that ExactOR, Majority and ExMaj do not have NC^0PS. Here, we prove that all these languages have DLTPS. In fact we prove slightly more: we show that for every function k computable in DLT, the language Exact_k has a DLTPS.

Before we start to give a general proof for arbitrary functions k, we will look at the specific case when $k(n) = 1$. So we need a proof system which outputs all strings with exactly one occurrence of 1, i.e. the language ExactOR.

By definition a proof system is a function $f : \Gamma^* \to \Sigma^*$, but in order to explain how a proof system works it is helpful to give some interpretation to the proof x, when outputting $f(x)$. In the case of ExactOR the proof should encode the position of the unique one in the string and the length of the output.

On (σ, i, x), the machine interprets the first $\log |x|$ bits of x as the prescribed position α for the unique 1 in the output string, and the length of x will be the length of the output. Let α denote the value of the first $\log |x|$ bits of x. The function computed by the machine is: $f(x) = 0^\alpha 1 0^{|x|-\alpha-1}$.

The machine can be described formally as follows:

$M(\sigma, i, x)$

```
Let $n = |x|$.
if $i < n$ then
  if $i = \alpha$ then
    If $\sigma = 1$ Accept.
  else
    If $\sigma = 0$ Accept.
  end if
end if
Reject.
```

$$x = \underbrace{\overbrace{\boxed{\alpha}}^{\log n}\ \dots\dots\dots\dots\dots}_{n}$$

Note that α need not be computed explicitly: to check whether $i = \alpha$, we only need to compare i with the first $\log |x|$ bits of x. Also note that $n = |x|$ can be computed in DLOGTIME. It is easy to see that the machine outputs strings with exactly one 1. As we cycle through all x, α gets all values in the range $[0, n-1]$. This ensures that the range of the function defined by the machine is ExactOR and it is onto.

▶ **Lemma 3.** ExactOR *has a* DLTPS.

Our next goal to prove the generalization for every function k computable in DLT. As a first step we give another simple proof system. Given a function $k : \mathbb{N} \to \mathbb{N}$ computable by a DLOGTIME Turing machine such that $\forall n : k(n) < n$, there is a DLTPS for the language $\{1^{k(n)} 0^{n-k(n)} \mid n \in \mathbb{N}\}$, i.e. the language has exactly one word of length n that consists of $k(n)$ ones followed by zeros. This language clearly has a DLTPS, nevertheless we will give the exact DLTPS which we will then extend to a proof system for Exact_k:

$M(\sigma, i, x)$

```
Let $n = |x|$.
if $i > n$ then
```

$$x = \underbrace{\dots\dots\dots\dots\dots\dots}_{n}$$

```
        Reject.
    else
        if $\sigma = 1$ and $i < k(n)$ then Accept.
        if $\sigma = 0$ and $i \geq k(n)$ then Accept.
        Reject.
    end if
```

Note that in the algorithm above we ignore all bits of x and only use the length of x to determine the output. The machine maps every input word of length n to the word $1^{k(n)}0^{n-k(n)}$. And therefore is the proof system for this language.

Every word in the output is already in Exact_k. However, Exact_k contains every permutation of these words too. To give a proof system for Exact_k, we will modify the above ensuring surjectivity. We interpret the input x as a list of numbers $\alpha_0, \dots, \alpha_{n-1}$ between 0 and $n-1$, where n is the length of the word we want to output. To access the i-th number in this list in this notation will require multiplication of s and i, where $s = \lceil \log n \rceil$. However, we do not know how to do this in DLOGTIME. Therefore, we store each number in DLOGTIME, as an s bit number where s is the smallest power of 2 greater of equal to $\log n$. Then we compute $s \cdot i$ by a simple bit shift in DLOGTIME. Also since we will allow arbitrary proofs, the number in the list might have a value larger than $n-1$, in this case we will interpret this number as the largest suitable number. Since these are only technical details we will simply write $\log n$ bit numbers in the rest of the paper.

Let $x = (\alpha_0 \dots \alpha_{n-1} \dots)$. We use the first n elements, i.e. $\alpha_0, \alpha_1, \dots, \alpha_{n-1}$, to come up with the output of length n and ignore the rest of the bits of x.

We interpret this list as a bijective map $m_{\vec{\alpha}}$. We say $m_{\vec{\alpha}}(i) = j$ if $\alpha_i = j$ and $\alpha_j = i$, if there is no such j we say $m_{\vec{\alpha}}(i) = i$. For every list α the map $m_{\vec{\alpha}}$ will be a bijection. Assume that $m(i) = j = m(i')$ then $i = \alpha_j = i'$ and hence m is injective. Since the map is between finite sets it is also surjective. Also note that every involution, i.e. a map m' such that $m'(m'(x)) = x$ for all x, can be represented in this way.

In order to have enough space in the proof for the whole list we will check that the proof has at least quadratic length l compared to the length n of string we want to output. Since we cannot exactly compute the square root we will approximate it. To approximate $\lfloor \sqrt{l} \rfloor$, let $j = \lfloor \log l \rfloor$ and let $k = \lfloor j/2 \rfloor$. As logarithm can be computed in DLOGTIME (see for example [1]) and division by 2 simply involves a shift by one bit to the right, j, k can be computed in DLOGTIME. Now, let n be the first j bits of the binary representation of l. Then n has the property $n^2 \leq l$, and for all $n \in \mathbb{N}$ there exists an $l \in \mathbb{N}$ such that n is the result of this operation. We define the function $\widehat{\text{sqrt}}(l) = n$, where n is obtained by the procedure above.

▶ **Theorem 4.** *For every function $k : \mathbb{N} \to \mathbb{N}$ computable in* DLT*,* Exact_k *has* DLTPS*.*

Proof. Consider the following proof system.

```
$M(\sigma, i, x)$

    Let $n = \widehat{\text{sqrt}}(|x|)$.
    if $i < n$ then
        Let $j = \alpha_i$
        if $\alpha_j = i$ then
            Let $t = j$.
        else
```

$$x = \underbrace{\overbrace{\boxed{\alpha_0}}^{\log n}\dots\overbrace{\boxed{\alpha_{n-1}}}^{\log n}\dots\dots\dots\dots}_{\approx n^2}$$

```
        Let $t = i$.
    end if
    if $\sigma = 1$ and $t < k(n)$ then Accept.
    if $\sigma = 0$ and $t \geq k(n)$ then Accept.
end if
Reject.
```

We now prove the correctness and surjectivity of M. In order to see that for a fixed x the output contains exactly $k(n)$ ones, note that we output a 1 if $t < k(n)$ and t is the image of i under a bijection. Given a word $w = w_0 \dots w_{n-1}$, we need to show that there is an x that produces the output w. Let I be the set of 1s in w. We will assign to each index in I a unique value β_i in the following way: Let $I_0 = \{i \in I \mid i < k(n)\}$. We define $\beta_i = i$ for $i \in I_0$ and for the remaining values of I are assigned the remaining numbers less than $k(n)$ in an arbitrary and one-to-one manner. The x which produces this w can now be described by specifying $\alpha_0, \alpha_1, \dots, \alpha_{n-1}$: $\alpha_i = \beta_i$ if $i \in I$, $\alpha_i = j$ if $\beta_j = i$, and $\alpha_i = i$ otherwise. It is easy to see that this input produces the output w. ◀

Since we can add additional 1s to the output as in [2], we get:

▶ **Corollary 5.** Majority *admits a* DLTPS.

Proof. As in [2] we can take a proof system of a language and add additional 1's to the output. Consider the proof system of Exact$_k$ where $k = \lfloor n/2 \rfloor + 1$. We extend the proof system by n additional bits $\gamma_1, \dots, \gamma_n$. Whenever would output a 0 at position i in the proof system of Exact$_k$ we will check if $\gamma_i = 1$, in which case we output a 1 instead. ◀

4 Problems on Graphs

In this section, we consider three problems on graphs. For every $n \in \mathbb{N}$, a directed graph on n vertices is in Reach if there exists a path from a vertex labelled 0 to a vertex labelled $n-1$ in the graph. For a fixed k, and for every $n \in \mathbb{N}$, a graph on n vertices is in Clique$_k$ if it has a clique of size at least k. For graph problems like Reach, Clique$_k$, the output of the proof system will be all graphs of these languages encoded as the adjacency matrix. The nodes of the graph are labeled by $1, \dots, n$ and hence the adjacency matrix has size $n \times n$. We assume that the positions of the words are indexed by (i, j) and $w_{(i,j)} = 1$ iff there is an edge from i to j.

If we were to index the positions by a single number k, we could use any DLOGTIME computable encoding. Though the usual encoding $k = i \cdot n + j$ is not immediately in DLOGTIME, a slight modification $k = i \cdot 2^{\lceil \log n \rceil} + j$ is computable in DLOGTIME. For such an encoding we would pad all positions not in the image of the tuple-function by a special character. In the following we use (i, j) to index the positions of the word.

▶ **Theorem 6.** Reach *has a* DLTPS

Proof. For an output of size $n \times n$ we require an input of length at most $n \log n + 2n^2 \leq 4n^2$. So we let $n = \widehat{\text{sqrt}}(|x|/4)$. We think of the beginning of x as a list of $\log n$-bit numbers $\alpha_0, \alpha_1, \dots, \alpha_{n-1}$. Also we pick any tuple function computable in DLOGTIME and think of the end of x as an $n \times n$ matrix of single bits β_{ij}. All other bits of x are ignored.

We interpret $\alpha = (\alpha_0, \alpha_1, \dots, \alpha_{n-1})$ as a path of length at most n, and $(\beta_{ij})_{i,j \in \{0,1,\dots,n-1\}}$ as a graph G. For an arbitrary input, we have no guarantee that the path α actually exists in the graph G. If the graph indeed has this path, we wish to preserve it in the output. On the other hand, if α is not a path in G, we wish to output a graph with a path. However, in the computation of a single output bit corresponding to the edge from i to j we cannot check whether α is a path in G, since the computation time is log-time bounded. So we spread this test among many output bits.

The smallest proof for existence of a path between 0 and $n-1$ is of constant size: for any i, adding edges $(0,i)$ and $(i,n-1)$ creates a positive instance. In DLT we can check for one i if α_i and α_{i+1} are the same nodes or they are connected in G. If they are not connected we will ensure that one of the short paths exists in the graph. If α indeed encodes a valid path, then we will simply copy G on the output tape. This way, we will generate all graphs that have paths from 0 to $n-1$.

It is easy to see that the following deterministic machine runs in $O(\log n)$ time. Observe that we output only positive instances of Reach. Suppose G is a graph with a path from 0 to $n-1$, then by repeating some of the vertices we get a sequence of nodes $(\alpha_0,\dots,\alpha_{n-1})$, which encodes this path. For this input, we will output exactly G. That is, for every positive instance of Reach, there is an input to the algorithm which outputs that instance.

$M(\sigma,(i,j),x)$

$$x = \underbrace{\overbrace{\boxed{\alpha_0}}^{\log n}\dots\overbrace{\boxed{\alpha_{n-1}}}^{\log n}\dots\overbrace{\boxed{\beta_{00}\dots\beta_{n-1,n-1}}}^{\leq 2n^2}}_{\approx 4n^2}$$

```
Let $n = \widehat{\mathrm{sqrt}}(|x|/4)$
if $i < n$ and $j < n$ then
  if $i = 0$ or ($i < n-1$ and $j = n-1$) then
    if $j = n-1$ then let $k = i$ else let $k = j$
    if $\beta_{\alpha_k,\alpha_{k+1}} \neq 1$ then
      {Check if $\alpha_k$ and $\alpha_{k+1}$ are connected}
      If $\sigma = 1$ Accept. {add the short path}
      If $\sigma = 0$ Reject.
    end if
  end if
  {No check needed or check succeeded, so output the graph $G$.}
  If $\sigma = \beta_{ij}$ Accept.
end if
Reject.
```

◀

▶ **Theorem 7.** *For any function $k:\mathbb{N}\to\mathbb{N}$ computable in* DLT*,* Clique$_k$ *is in* DLTPS

Proof. We begin with an easy case, where a graph with n nodes has a $k(n)$ clique among the nodes $0,\dots,k(n)-1$. The input consists of an arbitrary graph $G=(\beta_{ij})$. We output any graph in which if two vertices have labels less than $k(n)$ then they share an edge. All these graphs belong to Clique$_k$.

As in the previous proof we will use a bijection to permute the nodes (all the nodes including 0 and $n-1$ this time), which will give a correct and surjective proof system. The algorithm for Clique$_k$ can be found in the appendix.

◀

5 Closure Properties and Complexity

Closure under union seems to be naturally hold for all proof systems.

▶ **Lemma 8.** *If L_1, L_2 have* DLTPS*, then $L_1 \cup L_2$ has* DLTPS*.*

Proof. Let f_1, f_2 be the DLTPScorresponding to L_1, L_2. Then we define the function $f(x) = f_1(x_1 \ldots x_{n-1})$ if $x_0 = 0$, and $f(x) = f_2(x_1 \ldots x_{n-1})$ otherwise. Since f_1, f_2 are computable in DLT, so is f, and f is a proof system for $L_1 \cup L_2$.

The algorithm for union:

$M(\sigma, i, x)$

```
if $b = 0$ then
  Run $M_1(\sigma, i, x')$.
else
  Run $M_2(\sigma, i, x')$.
end if
```

$$x = \underbrace{\overbrace{\boxed{b}}^{1}\ \boxed{\quad\quad\quad x' \quad\quad\quad}}_{n}$$

◀

The closure under concatenation of proof systems is not so obvious.

▶ **Lemma 9.** *If L_1, L_2 have* DLTPS*, then L_1L_2 has* DLTPS*.*

Proof. By definition neither L_1 nor L_2 can be empty. Let $w_1 \in L_1$ and $w_2 \in L_2$. Let f_1, f_2 be the proof systems corresponding to L_1 and L_2, and M_1, M_2 the corresponding DLOGTIME machines.

We will construct a proof system for L_1L_2. The idea is that the proof consists of s, l, m, u, v, where s, l, m are of length $\log |x|$ and u, v are of length $|x|/4$. To concatenate $f_1(u_0 \ldots u_{l-1})$ and $f_2(v_0 \ldots v_{m-1})$, we will use s to "guess" the length of $f_1(u_0 \ldots u_{l-1})$.

Suppose we output the bit at position i. First we check if $|f_1(u_0 \ldots u_{l-1})| = s$, if this is not the case we output the i-th letter of the fixed default word w_1w_2. Otherwise if $i < s$ we output the i-th letter of $f_1(u_0 \ldots u_{l-1})$ and if $i \geq s$ we output the s-i-th letter of $f_2(v_0 \ldots v_{m-1})$.

Clearly the proof system will be correct since we only output words in L_1L_2. For surjectivity note that the input $u_0 \ldots u_{l-1}$ and $v_0 \ldots v_{m-1}$ are independent. Also since f_1, f_2 are polynomial bounded so is the constructed proof system. The algorithm for concatenation:

$M(\sigma, i, x)$

$$x = \underbrace{\overbrace{\boxed{s}}^{\log|x|}\ \overbrace{\boxed{l}}^{\log|x|}\ \overbrace{\boxed{m}}^{\log|x|}\ \ldots\ldots\ \overbrace{\boxed{\quad u \quad}}^{\lfloor|x|/4\rfloor}\ \overbrace{\boxed{\quad v \quad}}^{\lfloor|x|/4\rfloor}}_{|x|}$$

```
if $|f_1(u_0 \ldots u_{l-1})| = s$ then
  if $i < s$ then
    Run $M_1(\sigma, i, u_0 \ldots u_{l-1})$.
    {Accept or Reject depending on $M_1$}
  else
    Run $M_2(\sigma, i - s, v_0 \ldots v_{m-1})$.
    {Accept or Reject depending on $M_2$}
  end if
end if
{Output default word otherwise}
if $(w_1w_2)_i = \sigma$ then
  Accept.
else
  Reject.
```

```
    end if
```

In the algorithm we can test $|f_1(u_0 \dots u_{l-1})| = s$ by testing if there is a σ such that $M_1(\sigma, s-1, u_0 \dots u_{l-1})$ accepts and for all σ we have that $M_1(\sigma, s, u_0 \dots u_{l-1})$ rejects. This requires to simulate M_1 on $2|\Sigma|$ different inputs which is possible in logarithmic time. ◀

In [2], it was proved that every language L that has $\mathrm{NC}^0\mathrm{PS}$ is recognised in $\mathrm{NTIME}(n)$. In the case of $\mathrm{NC}^0\mathrm{PS}$ it is enough to guess a proof and then evaluate the circuit on this guessed proof. This suffices because the NC^0 circuit queries the proof bits in a non-adaptive manner. However, in the case of DLTPS the deterministic log-time machine may read bits on the proof in an adaptive manner, i.e. it may read a location, say i, of the proof and depending on the value of that proof bit, may decide to read the next bit. Therefore, the simulation of such a DLTPS needs to remain consistent with respect to such adaptive queries.

▶ **Theorem 10.** *If a language L has* DLTPS *then it can be recognised in NTIME$(n^2 \log^3 n)$.*

Proof. As L has a DLTPS there exists a $f : \Gamma^* \to \Sigma^*$ computable in DLT by a Turing machine M. Also since a DLTPS is polynomial bounded there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$, such that $w \in L$ iff $w \in f(\Sigma^{<p(|w|)})$.

Assume we guess the word x such that $w = f(x)$, then we could check that $M(\sigma, i, x)$ accepts iff $w_i = \sigma$. Since we want to show an upper bound of $n^2 \log^2 n$ we cannot simply guess x which might have size larger than $n^2 \log^2 n$.

But M is a DLT machine, so it cannot access all of the bits of x, but only $O(\log n)$ of these bits. Since we simulate M on $O(n)$ different inputs we require only $O(n \log n)$ of these bits. We guess and store only the bits of x accessed by M together with their indices on the tape which requires $O(n \log^2 n)$ length.

Then we can check that $f(x) = w$ by simulating M for all $i = 0, \dots, |w|$. For the simulation of a single step we might need to search on the tape of the bit accessed which requires $O(n \log^2 n)$ time. Since we simulate the machine $O(n)$ times with a runtime of $O(\log n)$ steps each, so we require time $O((n \log^2 n) \cdot n \cdot \log n)$. ◀

Using the non-deterministic time hierarchy theorem and Theorem 10 we get:

▶ **Corollary 11.** *There exists a language L in* NP *for which there is no* DLTPS.

We can use the previous theorem to show that DLTPS are not closed under intersection.

▶ **Theorem 12.** *The languages which have* DLTPS *are not closed under intersection.*

Proof. We show that if DLTPS are closed under intersection then all languages in NP have DLTPS. For this let $L \subset \Sigma^*$ be any problem in NP, i.e. in $\mathrm{NTIME}(n^c)$. We will construct two language L_a, L_b over the alphabet $\Sigma' = \Sigma \cup \{x, a, b\}$, such that both L_a, L_b have DLTPS and their intersection is L.

Let $L^{=n} = L \cap \Sigma^n$, and $B_a = \{x, a\}^* a \{x, a\}^*$, and $B_a^n = B_a \cap \Sigma^n$. Similar $B_b = \{x, b\}^* b \{x, b\}^*$, and $B_b^n = B_b \cap \Sigma^n$. Consider the languages $L_a = \bigcup_n L^{=n} x^{n^{2c}} \cup \Sigma^n B_a^{n^{2c}}$, $L_b = \bigcup_n L^{=n} x^{n^{2c}} \cup \Sigma^n B_b^{n^{2c}}$.

Each of the languages consists of each words of L padded by a certain number of xs, union some "bad" part which has at least one a for the first language or at least one b for the second language. If we take the intersection of $L_a \cap L_b$ we get the language L padded by some xs.

We will show that we have a proof system for L_a. So the proof will consist of the computation of the TM for L. This computation has at most length n^{2c}. While the

DLOGTIME for L_a outputs the input to the proof in the first n positions, we output at the remaining positions a x if the computation at this position is consistent, otherwise an a. Additionally we can output all words in $\Sigma^n B_a^{n^{2c}}$ and hence have a proof system for L_a.

Similarly we can construct a proof system for L_b. Assuming that DLTPS are closed under intersection we have a proof system for $L_a \cap L_b$. Assuming $x \notin \Sigma$, we can modify the DLOGTIME machine such that it always rejects when $\sigma = x$. This is only possible since no letter other than x can appear at a position behind x. (Otherwise the output would not be a word, but contains holes inside the word). The modified proof system is a proof system for L. Since L was any language in NP this is a contradiction. ◀

▶ **Corollary 13.** DLTPS *are not closed under complement.*

6 Hierarchy Theorems

▶ **Definition 14.** A function $f : \Gamma^* \to \Sigma^*$ is computable in DLT^k if there exists a deterministic Turing machine M such that $M(\sigma, i, x)$ halts and accepts in time $O((\log |x|)^k)$ if $w_i = \sigma$, where $w_0 \ldots w_{n-1} = f(x)$. We let DL$^{\mathrm{k}}$TPS be the polynomial bounded proof systems that are computable in DLT^k.

▶ **Theorem 15** (Time Hierarchy for Proof Systems). DL$^{\mathrm{t}}$TPS $\subsetneq$ DL$^{2\mathrm{t}+1}$TPS

Proof. The basic idea is to apply diagonalization as in the nondeterministic time hierarchy theorem. The machines that we consider are deterministic but we need to "guess" the proof.

We will only show the proof for DLTPS$\subsetneq$ DL^{3}TPS, the proof for any k is similar. The idea is to define a language $L \subseteq 1^*$ and show that L has a DL^{3}TPS but no DLTPS. Since we will be working over the unary alphabet we will ignore the parameter σ in our proof systems. Also it suffices in general to consider proofs in $\{0,1\}^*$. Let $M_0, M_1, M_2, \ldots$ be an enumeration of all DLOGTIME machines (which also includes DLOGTIME machines which are not DLTPS). The idea is to divide the natural numbers in intervals such that the left border of the interval is much smaller than the right border. For this we define a function $f : \mathbb{N} \to \mathbb{N}$ by:

$$f(0) = 2, f(l+1) = 2^{2^{2^{f(l)}}}$$

We will define the language L: Let k be any number and $t = \log^{1.1} k$.

1. L contains the empty word.
2. If $f(l) < k < f(l+1)$ for some l, then $1^k \in L$ iff there exists a word $w \in \{0,1\}^*$ of length less than or equal to 2^t such that $M_l(k, w)$ halts and accepts within t steps and $M_l(k+1, w)$ halts and rejects within t steps.
3. If $k = f(l+1)$ for some l, then $1^k \in L$ iff there exists a word $w \in \{0,1\}^*$ of length less than or equal to $2^{\log^{1.1} f(l)}$ such that $M_l(f(l), w)$ halts and rejects within t steps or $M_l(f(l)+1, w)$ halts and accepts within t steps.

For a DLOGTIME machine M_l, the definition basically says that:

1. If $f(l) < k < f(l+1)$ for some l, then $1^k \in L$ iff the proof system M_l outputs 1^{k+1}.
2. If $k = f(l+1)$ for some l, then $1^k \in L$ iff the proof system M_l does not output $1^{f(l)+1}$.

But keep in mind that this is not our definition, just the the intention for the definition.

We will show that if M_l is a DLTPS then the language corresponding to M_l is different from the language L. Since M_l is a DLOGTIME proof system we can assume that there is some c such that M_l halts after at most $c \log n$ steps. And since M_l is polynomial bounded, if there is a w such that M_l on input w will output 1^n there is a word w of length at most

$n^{c'}$. We assume that for all $n \geq f(l)$: $c \log n \leq \log^{1.1} n$ and $n^{c'} \leq 2^{\log^{1.1} n}$ (since there are infinite many l that represent the same DLOGTIME machine we can ensure this).

Assume by contradiction that the language of M_l equals L. By equality, we have that M_l outputs $1^{f(l)+1}$ iff L contains $1^{f(l)+1}$. By definition of L, for any k in $f(l) < k < f(l+1)$, 1^k is in L iff there exists a word $w \in \{0,1\}^*$ of length less or equal than $2^{\log^{1.1} k}$ such that $M_l(k, w)$ halts and accepts within t steps and $M_l(k+1, w)$ halts and rejects within t steps. Assuming that M_l is a proof system with the bounds on computation time and proof length as above, this happens iff M_l outputs 1^{k+1} (for some input). Hence by induction L contain $1^{f(l)+1}$ iff M_l outputs $1^{f(l+1)}$.

But by definition, L contains $1^{f(l+1)}$ iff M_l does not output $1^{f(l)+1}$ on any input (again by the assumption on the bounds of computation time and proof length). This is a contradiction.

$$\begin{tikzcd}
M_l(1^{f(l)+1}) \text{ acc} \arrow[d, Rightarrow] & M_l(1^{f(l)+2}) \text{ acc} \arrow[d, Rightarrow] & \dots & \dots & M_l(1^{f(l+1)}) \text{ acc} \arrow[d, Rightarrow] \\
1^{f(l)+1} \in L \arrow[ru, Rightarrow] & 1^{f(l)+2} \in L \arrow[ru, Rightarrow] & \dots & \dots & 1^{f(l+1)} \in L \arrow[llllu, dashed]
\end{tikzcd}$$

So L clearly has no DLTPS, it remains to show that there is a proof system D in DL^{3}TPS for L. We will construct a proof system D in DL^{3}TPS for L. The proof system will use a proof of length k to either output 1^k or an empty string.

For an input x we let $k_0 = |x|$ be the length of the input. We compute $k_{i+1} = \lfloor \log \log \log k_i \rfloor$, and repeat this process till we reach l such that $k_l = 0$. Then $f(l) < k \leq f(l+1)$, where equality occurs exactly if we never needed to round down somewhere in the process. We need at most $\log^* |x|$ repetitions, and except for the first repetition (which requires $O(\log n)$ steps) we only require $O(\log \log \log |x|)$ steps for the computation of one repetition, so clearly we can compute this in $O(\log n)$ steps.

Given an input x of length k such that there is an l with $f(l) < k < f(l+1)$. We want to output the word 1^k iff there exists a word w such that $M_l(k, w)$ halts and accepts within t steps and $M_l(k+1, w)$ halts and rejects within $t = \log^{1.1} k$ steps. First note that only two instance M_l are run each for t steps on w. Hence M_l will access only $2t$ positions of w. This implies that we can modify all bits other than these $2t$ positions without changing the acceptance behaviour of $M_l(k, w)$ and $M_l(k+1, w)$. In particular, we can set all positions other than these $2t$ positions to 0. Hence there is a word w such that $M_l(k, w)$ halts and accepts within t steps and $M_l(k+1, w)$ halts and rejects within t steps iff there is a word w with at most $2t$ bits set to 1 such that $M_l(k, w)$ halts and accepts within t steps and $M_l(k+1, w)$ halts and rejects within t steps.

This will greatly limit the amount of words w that we need to simulate. We interpret the proof x as a list of $\log^{1.1} |x|$ bit numbers: $s, \alpha_0, \ldots, \alpha_{2t}$ (we ignore all other bits). We will let $w = \phi(x)$ where $\phi(x)$ is a word of length s that has a bit set to 1 at the positions $\alpha_0, \ldots, \alpha_{2t}$. Then there is a word x of length k such that $M_l(k, \phi(x))$ halts and accepts within t steps and $M_l(k+1, \phi(x))$ halts and rejects within t steps iff there exists a word $w \in \{0,1\}^*$ of length less or equal than 2^t such that $M_l(k, w)$ halts and accepts within t steps and $M_l(k+1, w)$ halts and rejects within t steps. Hence for our proof system it suffices to simulate $M_l(k, \phi(x))$ and $M_l(k+1, \phi(x))$ for t steps, and output 1^k accordingly to the definition of L.

Now assume that $k = f(l+1)$, then the proof system on an input x of length k should output 1^k iff there exists a word $w \in \{0,1\}^*$ of length less or equal than $2^{\log^{1.1}(f(l)+1)}$ such that $M_l(f(l), w)$ halts and rejects within t steps or $M_l(f(l)+1, w)$ halts and accepts within t steps. So here we need to check if M_l outputs a word of length $f(l)+1$, where $f(l)$ is much smaller than $f(l+1)$. That is, the simulating machine runs for $\log^{1.1}(f(l)+1)$ steps and for each word of length at most $2^{\log^{1.1}(f(l)+1)}$. Therefore, the total running time of the simulating machine is upper bounded by $2^{2^{f(l)}}$. As the input length (as well as the output length) in this case is $2^{2^{2^{f(l)}}}$, our machine has enough time to actually simulate all possible words and check the output of M_l (in this case M_l ignores the bits of x, we only require a long x to have sufficient time for the simulation).

This completes the proof showing that L has a DL^3TPS (in fact $\mathrm{DL}^{2+\varepsilon}$TPS). ◀

Acknowledgements We thank Klaus-Jörn Lange for helpful comments on this draft.

References

1 David A. Mix Barrington, Neil Immerman, and Howard Straubing. On Uniformity within NC^1. *J. Comput. Syst. Sci.*, 41(3):274–306, 1990.

2 Olaf Beyersdorff, Samir Datta, Andreas Krebs, Meena Mahajan, Gido Scharfenberger-Fabian, Karteek Sreenivasaiah, Michael Thomas, and Heribert Vollmer. Verifying proofs in constant depth. In Filip Murlak and Piotr Sankowski, editors, *Full version on ECCC TR12-079, a preliminary version in MFCS*, volume 6907 of *Lecture Notes in Computer Science*, pages 84–95. Springer, 2011.

3 Olaf Beyersdorff, Johannes Köbler, and Sebastian Müller. Proof systems that take advice. *Inf. Comput.*, 209(3):320–332, 2011.

4 Olaf Beyersdorff and Sebastian Müller. A tight karp-lipton collapse result in bounded arithmetic. In Michael Kaminski and Simone Martini, editors, *Computer Science Logic*, volume 5213 of *Lecture Notes in Computer Science*, pages 199–214. Springer Berlin Heidelberg, 2008.

5 Stephen A. Cook and Jan Krajícek. Consequences of the provability of NP subset of or equal to P/poly. *J. Symb. Log.*, 72(4):1353–1371, 2007.

6 Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *The Journal of Symbolic Logic*, 44(1):pp. 36–50, 1979.

7 Birgit Jenner, Pierre McKenzie, and Jacobo Torán. A note on the hardness of tree isomorphism. In *IEEE Conference on Computational Complexity*, pages 101–105, 1998.

8 Andreas Krebs, Nutan Limaye, Meena Mahajan, and Karteek Sreenivasaiah and. Small depth proof systems. In *a preliminary version to appear in MFCS*, 2013.

9 Pavel Pudlák. Quantum deduction rules. *Annals of Pure and Applied Logic*, 157(1):16 – 29, 2009.

10 Stanislav Žák. A turing machine time hierarchy. *Theoretical Computer Science*, 26(3):327 – 333, 1983.

A DLOGTIME proof system for Clique_k

DLTPS proof system for Clique_k is described below:

$M(\sigma, (i, j), x)$

$$x = \underbrace{\overbrace{\boxed{\alpha_0}}^{\log n} \dots \overbrace{\boxed{\alpha_{n-1}}}^{\log n} \dots \overbrace{\boxed{\beta_{00} \dots \beta_{n-1,n-1}}}^{\leq 2n^2}}_{\approx 4n^2}$$

```
Let $n = \widehat{\mathrm{sqrt}}(|x|/4)$.
if $i < n$ then
  Let $k = \alpha_i$. {Find the image $i'$ of $i$}
  if $\alpha_k = i$ then
    Let $i' = k$.
  else
    Let $i' = i$.
  end if
  Let $l = \alpha_j$. {Find the image $j'$ of $j$}
  if $\alpha_l = j$ then
    Let $j' = l$.
  else
    Let $j' = j$.
  end if
  if $i' < s(n)$ and $j' < s(n)$ then
    If $\sigma = 1$ accept
    Otherwise reject.
  else
    If $\sigma = \beta_{i',j'}$ accept,
    otherwise reject.
  end if
end if
Reject.
```